TEACHING OLD COMPILERS NEW TRICKS

Transpiling C++17 to C++11

Tony Wasserka
@fail_cluez

C++ on Sea
5 February 2019

WHO AM I?

- C++ enthusiast: Systems programming & Type-safety
- Game console emulation

PPSSPP

Dolphin

Citra

- Consulting: Code Modernization, reviews, etc

C++14/17: WHY?

```
auto [it, inserted] = my_map.insert(...);
```

```
uint32_t bitmask = 0b1000'0010;
```

```
if (QVariant v = getAnswer(); v.isValid())
   use(var);
```

```
optional<Token> parse(string&);
```

```
using calc_expr = variant<sum, prod>;
calc_expr parsed = calc_parse("5+3*8");
std::visit(eval_expr, parsed);
```

```
template<typename... Bars>
auto total_foos(Bars... bars) {
    return (bars.foo() + ...);
}
```

```
if constexpr (is_array_v<T>) {
  return t[5];
} else {
  return t;
}
```

```
path dir = temp_directory_path() / "test";
create_directory(dir);
```

THE PROBLEM

C++17

```
int get_mask() {
  return 0b1000'0000;
}
```

gcc 4.9 →

Assembly

```
get_mask:
  mov eax, 128
  ret
```

C++17

```
int get_mask() {
  return 0b1000'0000;
}
```

gcc 4.8 →

Assembly

```
get_mask:
  mov eax, 128
  ret
```

SOLUTIONS

- Upgrade the compiler…
  - … if you can
- Stick with old C++…
  - …and put up with the consequences
- Teach your compiler some C++17

Enter Clang-from-the-Future

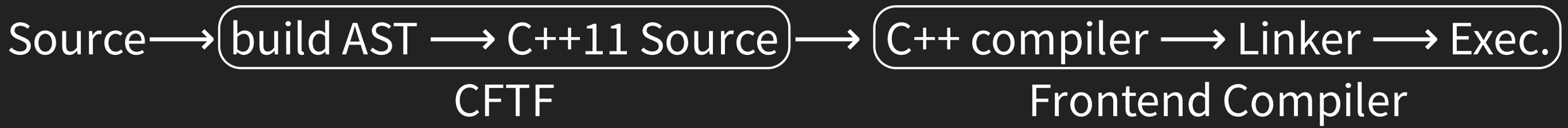
CLANG-FROM-THE-FUTURE

- libclang-based tool
- Preprocessor before the compiler runs

Source ⟶ build AST ⟶ C++11 source ⟶ C++ compiler ⟶ Linker ⟶ Exec.

- Automatic conversion of C++17 to C++11

CFTF-ENHANCED PIPELINE



CFTF = Black box precompilation step

```
int get_mask() {
  return 0b1000'0000;
}
```

```
int get_mask() {
  return 128;
}
```

A SOLUTION

C++17

```
int get_mask() {
  return 0b1000'0000;
}
```

gcc 4.9 →

Assembly

```
get_mask:
  mov eax, 128
  ret
```

C++17

```
int get_mask() {
  return 0b1000'0000;
}
```

gcc 4.8 →

Assembly

~~get_mask: mov eax, 128 ret~~

C++17

```
int get_mask() {
  return 0b1000'0000;
}
```

CFTF →

C++11

```
int get_mask() {
  return 128;
}
```

gcc 4.8 →

Assembly

```
get_mask:
  mov eax, 128
  ret
```

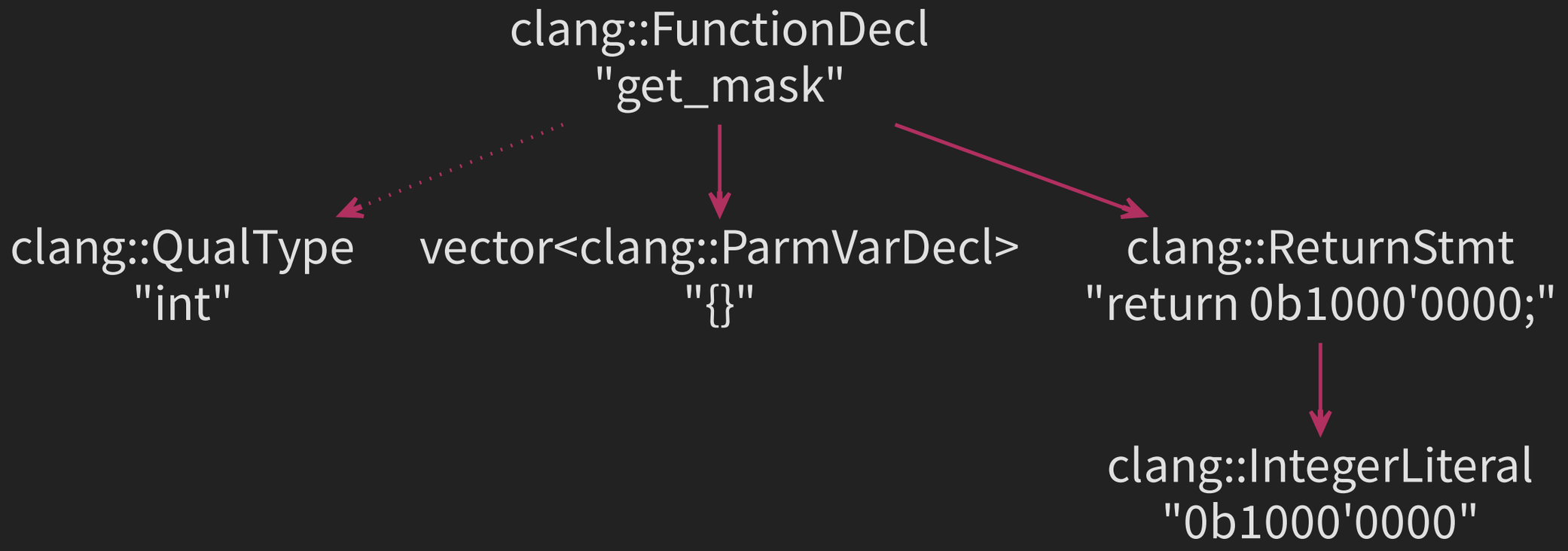
HOW IT WORKS

THE CLANG TOOLING UNIVERSE

- clang-format
- clang-tidy
- CPP2C
- C++ Insights
- Reflection frameworks
- Static analysis
- Tons of custom tools

ASTs: Abstract Syntax Trees

```
int get_mask() {
  return 0b1000'0000;
}
```

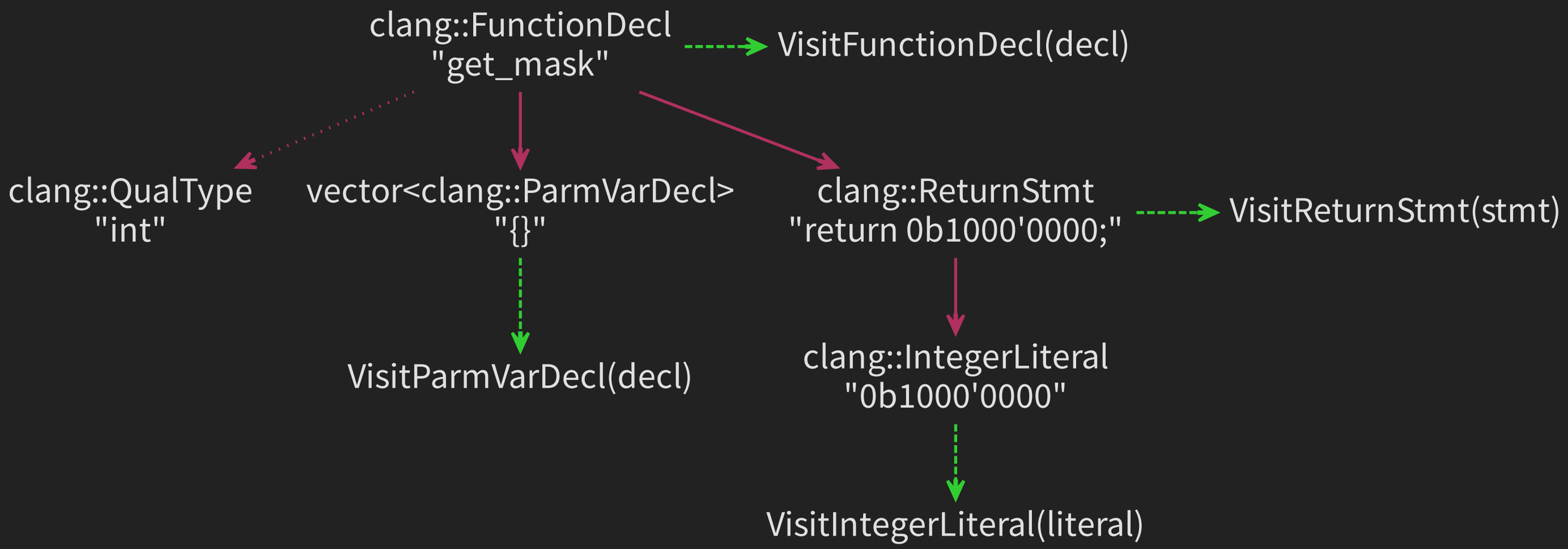


Dump using

```
clang -Xclang -ast-dump file.cpp
```

AST VISITORS

Call C++ function for each node type



Handled via clang::RecursiveASTVisitor

AST VISITORS

```
void MyASTVisitor::VisitIntegerLiteral(clang::IntegerLiteral* literal) {
  llvm::APInt value = literal->getValue();
  rewriter->ReplaceText(literal->getSourceRange(), value.toString());
}
```

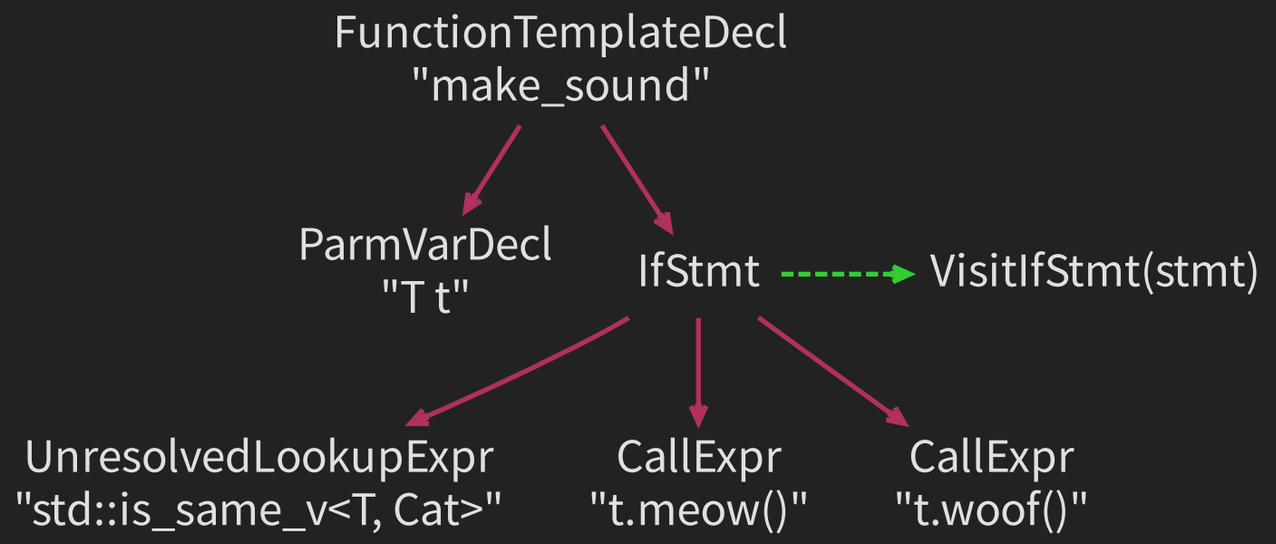
```
int get_mask() {
  return 0b1000'0000;
}
```

```
int get_mask() {
  return 128;
}
```

“Cool, now do constexpr if!”

CONSTEXPR IF

```
template<typename T>
void make_sound(T t) {
  if constexpr (std::is_same_v<T, Cat>) {
    t.meow();
  } else {
    t.woof();
  }
}
```



 So which branch is it?

DEPENDENT CONTEXTS

```
template<typename T>
void make_sound(T t) {
  if constexpr (std::is_same_v<T, Cat>) {
    t.meow();
  } else {
    t.woof();
  }
}
```

```
template<>
void make_sound(Cat t) {
  using T = Cat;
  if (std::is_same_v<T, Cat>) {
    t.meow();
  } else {
    // Nothing
  }
}
```

```
template<>
void make_sound(Dog t) {
  using T = Dog;
  if (std::is_same_v<T, Cat>) {
    // Nothing
  } else {
    t.woof();
  }
}
```

Template specializer: Implicit ⇒ Explicit instantiations

CONSTEXPR IF

```
void MyASTVisitor::VisitIfStmt(clang::IfStmt* stmt) {
  if (!stmt->isConstexpr())
    return;

  clang::Expr* cond = stmt->getCond(); // e.g. "std::is_same_v<T, Cat>"
  bool result;
  cond->EvaluateAsBooleanCondition(result, context);
  clang::Stmt* branch_taken = result ? stmt->getThen() : stmt->getElse();

  // Remove "constexpr"
  rewriter->ReplaceText(stmt->getLocStart(), cond->getLocStart(), "if (");

  // Remove inactive branch body
  if (!result) {
    rewriter->ReplaceText(cond->getLocEnd(), branch_taken->getLocStart(), ") {} else");
  } else {
    rewriter->ReplaceText(branch_taken->getLocEnd(), stmt->getLocEnd(), "");
  }
}
```

IN PRACTICE

```
cftf  -frontend-compiler=g++  input.cpp
```

Easy integration into your build pipeline:

- Make:

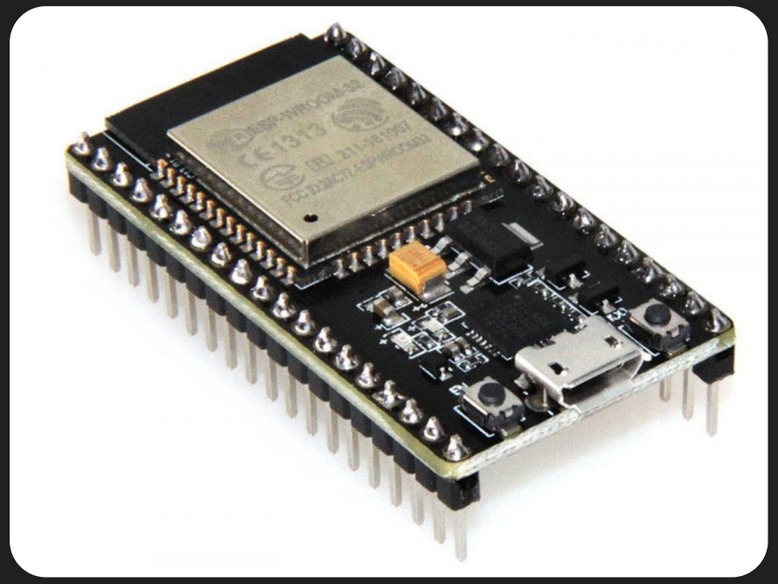
  ```
  CXX=cftf CXX_FLAGS="-frontend-compiler=g++" make
  ```

- CMake:

  ```
  CXX=cftf cmake -DCMAKE_CXX_FLAGS="-frontend-compiler=g++" .
  ```

- Other setups may need some creativity

SHOWCASE



USE CASES

- Early adoption of new standards
- Use of C++17 libraries in C++11 setups
- Ports to legacy platforms
- Cherry-pick specific features: Concepts, Contracts

CURRENT STATUS

- Usable drop-in for gcc/clang on Linux
  Windows/macOS support planned!
- Small initial set of supported C++14/17 features
  (correctness first, then features)

Available for licensing now

Prototype on GitHub

(Full version is proprietary)

FUTURE

- Better debugging experience
- More test cases
- C++11 to C++03
- C++20: Contracts, concepts, …
- Robustness against macros
- Platform integration: Visual Studio, Xcode

Your wishes?

Let's talk!

SUMMARY

Clang-from-the-Future:

- Compile C++14/17 on an old compiler
- Functional drop-in preprocessor
- Easy integration into existing toolchains
- No source code changes needed

github.com/neobrain/cftf

@fail_cluez tony.wasserka@gmx.de neobrain